ORYX

ORYX

- ORYX not an acronym, but upper case
- Designed for use with cell phones
  - To protect confidentiality of voice/data
  - For "data channel", not "control channel"
  - Control channel encrypted with CMEA
- Standard developed by
  - Telecommunications Industry Association (TIA)
  - Flaws in cipher discovered in 1997
- Cipher design process not open
  - In violation of Kerckhoffs Principle

Stream Cipher

- ORYX is a stream cipher
- Recall that a stream cipher can be viewed as a generalization of one-time pad
- A stream cipher initialized with (short) **key**
- Cipher then generates a long **keystream**
- Keystream is used like a one-time pad
  - XOR with message to encrypt/decrypt

ORYX

- Stream cipher
- Uses 3 shift registers, denoted X, A, B
  - Each holds 32 bits
- Key is initial fill of registers
- Therefore, ORYX has 96 bit key
- ORYX also uses a lookup table L
  - Where L acts as IV (or MI)
- Note that L is **not** secret

ORYX

- ORYX generates keystream 1 byte/step
- Most stream ciphers generate 1 bit/step
  - RC4 stream cipher also generates bytes
- ORYX is very weak
  - RC4 is weak due to the way it is used in WEP
- What is wrong with ORYX?
  - Generates **1 byte** of keystream
  - Too much of the internal state is exposed
  - Might be stronger if only generated **1 bit**/step

ORYX Shift Registers

- Three shift registers, X, A, B, each 32 bits
- Feedback functions (polynomials)
  - Register X uses feedback polynomial

    $P_X = x_0 \oplus x_4 \oplus x_5 \oplus x_8 \oplus x_9 \oplus x_{10} \oplus x_{13} \oplus x_{15} \oplus x_{17} \oplus x_{18} \oplus x_{27} \oplus x_{31}$
  - Register A uses two polynomial

    $P_{A0} = a_0 \oplus a_1 \oplus a_3 \oplus a_4 \oplus a_6 \oplus a_7 \oplus a_9 \oplus a_{10} \oplus a_{11} \oplus a_{15} \oplus a_{21} \oplus a_{22} \oplus a_{25} \oplus a_{31}$

    $P_{A1} = a_0 \oplus a_1 \oplus a_6 \oplus a_7 \oplus a_8 \oplus a_9 \oplus a_{10} \oplus a_{12} \oplus a_{16} \oplus a_{21} \oplus a_{22} \oplus a_{23} \oplus a_{24} \oplus a_{25} \oplus a_{26} \oplus a_{31}$
  - Register B uses polynomial

    $P_B = b_0 \oplus b_2 \oplus b_5 \oplus b_{14} \oplus b_{15} \oplus b_{19} \oplus b_{20} \oplus b_{30} \oplus b_{31}$

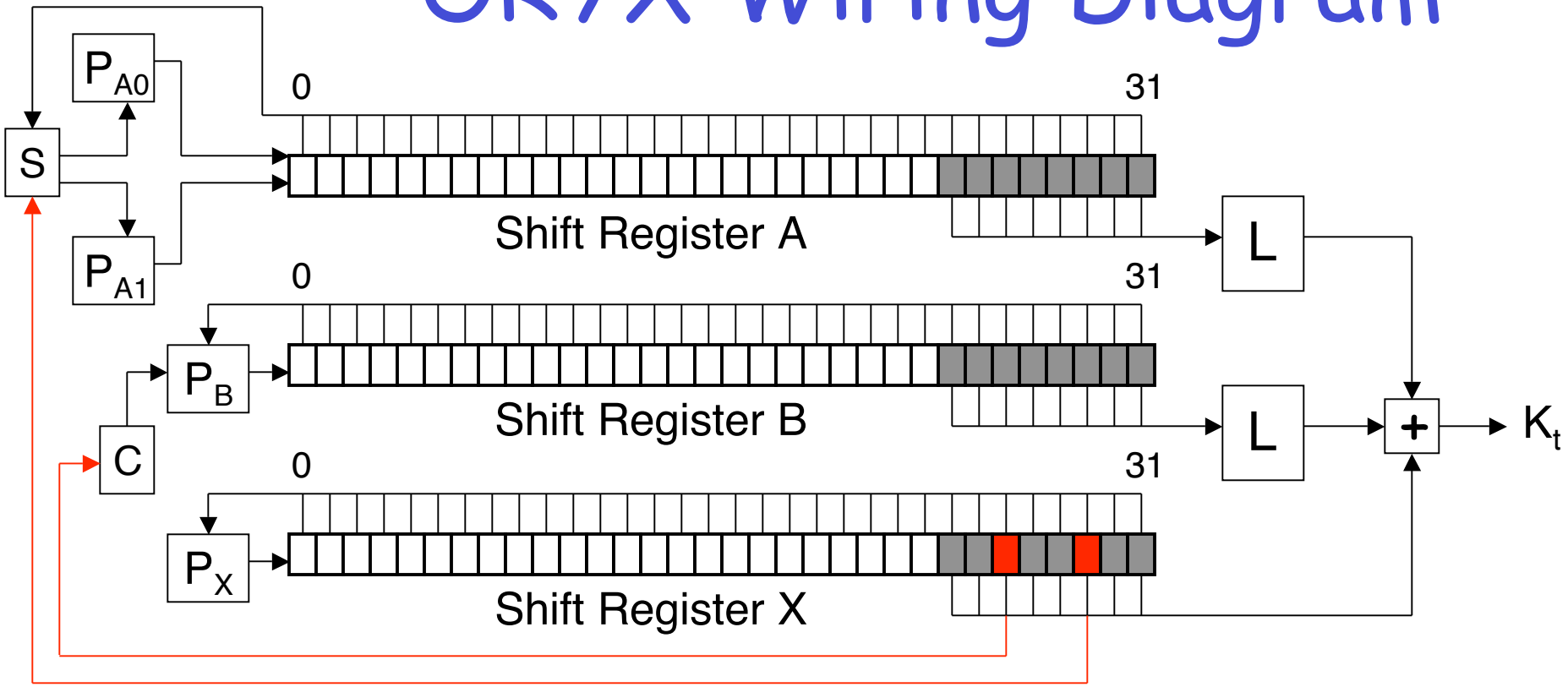
ORYX Lookup Table

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	ed	3e	0d	20	a9	c3	36	75	4c	2c	57	a3	00	ae	31	0f
1	19	4d	44	a0	11	56	18	66	09	69	6e	3d	25	9c	db	3f
2	65	58	1a	6d	ff	d7	46	b3	b1	2b	78	cf	be	26	42	2f
3	d8	d4	8e	48	05	b9	34	43	de	68	5a	aa	9d	bd	84	a2
4	3c	50	ce	8b	c5	d0	a5	77	1f	12	6b	c2	b5	e6	ab	54
5	81	22	9f	bb	5c	a8	dc	ec	2d	1e	ee	d6	6c	5f	9a	fd
6	c8	d5	94	fc	0c	1c	96	4f	f9	51	da	9b	df	e1	47	37
7	d1	eb	af	f7	a4	03	f0	c7	60	e4	f4	b4	85	f6	62	04
8	71	87	ea	17	99	1d	3a	15	52	0a	07	35	e0	70	b6	fa
9	cb	b0	86	a6	92	fb	98	55	06	4b	5d	4a	45	83	bf	16
a	7c	10	95	28	38	82	f3	6a	f8	fe	79	39	27	2a	5e	e7
b	59	b8	1b	ca	8d	d3	7b	30	33	90	d2	d9	ac	76	8f	5b
c	a7	0e	63	c4	b2	e9	97	91	53	7a	0b	41	08	c1	8c	7d
d	88	24	f5	f2	01	72	e8	80	49	13	23	9e	c6	14	73	ad
e	8a	29	ef	e5	67	61	ba	e2	7e	89	64	02	c0	21	6f	f1
f	dd	b7	c9	e3	cd	3b	93	2e	40	bc	4e	a1	cc	74	32	7f

- Example lookup table L
- Table is not secret

- Table L is different for each message
  - 1 byte in, 1 byte out
- Consider L to left
  - L[5a] = ee
    - Row 5, column a
  - L[d2] = f5
    - Row d, column 2

ORYX Wiring Diagram



- S determines whether to use P_{A0} or P_{A1}
- C determines whether B steps 1 or 2 times
- L is a fixed (per msg), known lookup table

ORYX Keystream Generator

1. Polynomial X steps once using P_X
2. Polynomial A steps once using P_{A0} or P_{A1}
   As determined by bit 29 of X
3. Polynomial B steps 1 or 2 times using P_B
   As determined by bit 26 of X
4. Byte: $k_i = H(X) + L[H(A)] + L[H(B)] \bmod 256$
   Where L is a known lookup table and
   H is "high" byte, i.e., bits 24 through 31
   k_i is i^{th} keystream byte

Note: Polynomials step **before** generating byte

ORYX Keystream

- Let $k_0, k_1, k_2, \ldots$ be keystream bytes
- Then k_0 given by

  $k_0 = H(X) + L[H(A)] + L[H(B)] \bmod 256$

  where X, A, B are fills **after 1st iteration**
- Then k_1 given by

  $k_1 = H(X) + L[H(A)] + L[H(B)] \bmod 256$

  where X, A, B are fills **after 2nd iteration**
- And so on…

ORYX Attack

- Attack is not difficult
- Idea of the attack
  - Suppose we know plaintext bytes $p_0,p_1,\ldots,p_n$
  - Then we know keystream bytes, $k_0,k_1,\ldots,k_n$
  - At each iteration, small change in internal state
  - Byte of output gives lots of info on state
  - Given enough plaintext, reconstruct initial fills
- Only need small amount of known plaintext

ORYX Attack

j	shift A	shift B	extend fill A	extend fill B
0	1	1	0	0
1	1	1	0	1
2	1	1	1	0
3	1	1	1	1
4	1	2	1	00
5	1	2	1	01
6	1	2	1	10
7	1	2	1	11
8	1	2	0	00
9	1	2	0	01
10	1	2	0	10
11	1	2	0	11

- Given register fills A,B
- Fill A can be extended in 2 ways: 0 or 1
- Fill B can be extended 6 ways: 0,1,00,01,10,11
- Table lists possible extensions of A,B
- New bit(s) appear on **left** of H(A),H(B)

ORYX Attack

- Given keystream bytes k_i where

  $k_i = H(X) + L[H(A)] + L[H(B)] \bmod 256$

- And X, A, B are fills **after** (i+1)st iteration
- For each of 2^{16} possible (H(A),H(B))
  - Let $H(X) = k_0 - L[H(A)] - L[H(B)] \bmod 256$
  - For each of 12 extensions of (A,B)
  - Let $Y = k_1 - L[H(A)] - L[H(B)] \bmod 256$
  - Is Y a possible extension of H(X) ?
    - If yes, save result for testing with bytes k_2, k_3, …
  - If no valid extension, discard (H(A),H(B))

ORYX Attack Example

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	ed	3e	0d	20	a9	c3	36	75	4c	2c	57	a3	00	ae	31	0f
1	19	4d	44	a0	11	56	18	66	09	69	6e	3d	25	9c	db	3f
2	65	58	1a	6d	ff	d7	46	b3	b1	2b	78	cf	be	26	42	2f
3	d8	d4	8e	48	05	b9	34	43	de	68	5a	aa	9d	bd	84	a2
4	3c	50	ce	8b	c5	d0	a5	77	1f	12	6b	c2	b5	e6	ab	54
5	81	22	9f	bb	5c	a8	dc	ec	2d	1e	ee	d6	6c	5f	9a	fd
6	c8	d5	94	fc	0c	1c	96	4f	f9	51	da	9b	df	e1	47	37
7	d1	eb	af	f7	a4	03	f0	c7	60	e4	f4	b4	85	f6	62	04
8	71	87	ea	17	99	1d	3a	15	52	0a	07	35	e0	70	b6	fa
9	cb	b0	86	a6	92	fb	98	55	06	4b	5d	4a	45	83	bf	16
a	7c	10	95	28	38	82	f3	6a	f8	fe	79	39	27	2a	5e	e7
b	59	b8	1b	ca	8d	d3	7b	30	33	90	d2	d9	ac	76	8f	5b
c	a7	0e	63	c4	b2	e9	97	91	53	7a	0b	41	08	c1	8c	7d
d	88	24	f5	f2	01	72	e8	80	49	13	23	9e	c6	14	73	ad
e	8a	29	ef	e5	67	61	ba	e2	7e	89	64	02	c0	21	6f	f1
f	dd	b7	c9	e3	cd	3b	93	2e	40	bc	4e	a1	cc	74	32	7f

- Spse k_0 = da, k_1 = 31
- Consider case where
  H(A) = b3, H(B) = 84
- Then, modulo 256,
  H(X) = da – L[b3] – L[84]
  H(X) = da – ca – 99 = 77

- Can we extend H(X) = 77 to next iteration?
- Must use k_1 and consider all 12 cases

ORYX Attack Example

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	ed	3e	0d	20	a9	c3	36	75	4c	2c	57	a3	00	ae	31	0f
1	19	4d	44	a0	11	56	18	66	09	69	6e	3d	25	9c	db	3f
2	65	58	1a	6d	ff	d7	46	b3	b1	2b	78	cf	be	26	42	2f
3	d8	d4	8e	48	05	b9	34	43	de	68	5a	aa	9d	bd	84	a2
4	3c	50	ce	8b	c5	d0	a5	77	1f	12	6b	c2	b5	e6	ab	54
5	81	22	9f	bb	5c	a8	dc	ec	2d	1e	ee	d6	6c	5f	9a	fd
6	c8	d5	94	fc	0c	1c	96	4f	f9	51	da	9b	df	e1	47	37
7	d1	eb	af	f7	a4	03	f0	c7	60	e4	f4	b4	85	f6	62	04
8	71	87	ea	17	99	1d	3a	15	52	0a	07	35	e0	70	b6	fa
9	cb	b0	86	a6	92	fb	98	55	06	4b	5d	4a	45	83	bf	16
a	7c	10	95	28	38	82	f3	6a	f8	fe	79	39	27	2a	5e	e7
b	59	b8	1b	ca	8d	d3	7b	30	33	90	d2	d9	ac	76	8f	5b
c	a7	0e	63	c4	b2	e9	97	91	53	7a	0b	41	08	c1	8c	7d
d	88	24	f5	f2	01	72	e8	80	49	13	23	9e	c6	14	73	ad
e	8a	29	ef	e5	67	61	ba	e2	7e	89	64	02	c0	21	6f	f1
f	dd	b7	c9	e3	cd	3b	93	2e	40	bc	4e	a1	cc	74	32	7f

- Have: $k_1 = 31$, H(A) = b3, H(B) = 84, H(X) = 77
- Try extension where

  A→1A, B→0B
- Then

  Y = 31 – L[d9] – L[42]

  Y = 31 – 13 – ce = 50

- But H(X) = 77 can only be extended to 3b or bb
- So this is **not** a valid extension
- Must consider 11 more cases for (H(A),H(B)) pair

ORYX Attack Example

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	ed	3e	0d	20	a9	c3	36	75	4c	2c	57	a3	00	ae	31	0f
1	19	4d	44	a0	11	56	18	66	09	69	6e	3d	25	9c	db	3f
2	65	58	1a	6d	ff	d7	46	b3	b1	2b	78	cf	be	26	42	2f
3	d8	d4	8e	48	05	b9	34	43	de	68	5a	aa	9d	bd	84	a2
4	3c	50	ce	8b	c5	d0	a5	77	1f	12	6b	c2	b5	e6	ab	54
5	81	22	9f	bb	5c	a8	dc	ec	2d	1e	ee	d6	6c	5f	9a	fd
6	c8	d5	94	fc	0c	1c	96	4f	f9	51	da	9b	df	e1	47	37
7	d1	eb	af	f7	a4	03	f0	c7	60	e4	f4	b4	85	f6	62	04
8	71	87	ea	17	99	1d	3a	15	52	0a	07	35	e0	70	b6	fa
9	cb	b0	86	a6	92	fb	98	55	06	4b	5d	4a	45	83	bf	16
a	7c	10	95	28	38	82	f3	6a	f8	fe	79	39	27	2a	5e	e7
b	59	b8	1b	ca	8d	d3	7b	30	33	90	d2	d9	ac	76	8f	5b
c	a7	0e	63	c4	b2	e9	97	91	53	7a	0b	41	08	c1	8c	7d
d	88	24	f5	f2	01	72	e8	80	49	13	23	9e	c6	14	73	ad
e	8a	29	ef	e5	67	61	ba	e2	7e	89	64	02	c0	21	6f	f1
f	dd	b7	c9	e3	cd	3b	93	2e	40	bc	4e	a1	cc	74	32	7f

- Have: $k_1 = 31$, H(A) = b3, H(B) = 84, H(X) = 77
- Try extension where

  A→0A, B→00B
- Then

  Y = 31 – L[59] – L[21]

  Y = 31 – 1e – 58 = bb

- Note that H(X) = 77 can be extended to bb
- So this is a **consistent** extension of H(X)
- Save it, and consider remaining extensions

ORYX Attack

- Attack algorithm
  - We have described a breadth first search
  - Depth-first search would work too
- But is the attack practical?
  - Can we really determine correct initial fill?
  - How efficient is attack?
  - How much known plaintext is required?
- We can easily answer these questions…

ORYX Attack

- Guess all pairs of initial (H(A),H(B))
  - There are $2^{16} = 65536$ of these
  - For each, use k_0 to solve for H(X)
- Have 2^{16} putative triples (H(A),H(B),H(X))
- For each of these 65536
  - For each of 12 extensions of (H(A),H(B)), compute $Y = k_1 - L[H(A)] - L[H(B)] \bmod 256$
  - Note that Y is putative extension of X
  - So, 7 bits of Y must agree with 7 bits from H(X)
- Expect $(12 \cdot 65536) / 128 = 6144$ survivors

ORYX Attack

- Expect 6144 survivors using k_1
- For each of these 6144, use k_2 and repeat the process…
  - Compute 12 extensions of (H(A),H(B))
  - New Y must agree with 7 bits of putative X (previous Y)
  - Expect $(12 \cdot 6144) / 128 = 576$ survivors
- Expected number of survivors decreases by factor greater that 10 at each step
  - Eventually, only 1 survivor is expected

ORYX Attack

- After 6 steps, expect only 1 survivor
- Keep going…
- After about 25 keystream bytes, X,A,B fills will be completely known
  - Not initial fills, but fills after 1st iteration
  - Easy to "back up" to initial fills, if desired
- Suppose n keystream bytes used
- Then work is on the order of

$$12 \cdot (65536 + 6144 + 576 + 54 + n) < 2^{20}$$

ORYX

- ORYX is **very insecure**
  - Work of about 2^{20} to recover 96-bit key!
- What is the problem?
  - One iteration does not change internal state very much
  - Byte of keystream gives lots of info on state
- Can it be improved?
  - Many alternatives…

More Secure ORYX ?

- ORYX does the following

  byte = H(X) + L[H(A)] + L[H(B)] mod 256

  where H is high-order byte
- Could instead do

  bit = s(X) $\oplus$ s(L[H(A)]) $\oplus$ s(L[H(B)])

  where s selects the high-order **bit** of a byte
- Then previous attack does not work, since
  - Expect $(12 \cdot 2^{16}) / 1 = 2^{19.6}$ after k_1
  - Expect $(12 \cdot 2^{19.6}) / 1 = 2^{23.2}$ after k_2, etc.
- But this "secure" ORYX is very slow!
- And may be other attacks to worry about…